Java ING1

Examen sur machine

Tous documents autorisés

Problème les 100 prisonniers

Il s'agit de modéliser un problème assez étonnant appelé le problème des 100 prisonniers.

100 prisonniers sont condamnés à mort. Le directeur de la prison propose un challenge à nos prisonniers

- il leur attribue à tous un numéro entre 1 et 100
- il installe dans son bureau une armoire avec 100 tiroirs, dans chacun desquels il met aléatoirement un et un seul numéro entre 1 et 100. Chaque numéro apparaît donc une et une seule fois.

Il propose à chaque prisonnier de venir ouvrir 50 tiroirs de son bureau, pour regarder le numéro qui est dedans. Les prisonniers sont d'abord réunis pour élaborer une stratégie puis envoyer dans un ordre aléatoire dans le bureau. Une fois passés dans le bureau, les prisonniers ne peuvent pas communiquer entre eux, ni changer les numéros de place, ni laisser un tiroir ouvert, ni coller un chewing-gum sur l'interrupteur de la lampe... Ils ne verront jamais les autres prisonniers avant le jugement dernier.

De deux choses l'une :

- Tous les prisonniers ont trouvé leur **numéro** en ouvrant les tiroirs auxquels ils avaient droit : ils sont tous graciés.
- Sinon, ils sont tous exécutés.

Un probabiliste dans le groupe des prisonniers dit : "aie aie aie ! On est mal : 1 chance sur 2^100 de s'en sortir". A-t-il vraiment raison ? N'y a-t-il pas un moyen d'augmenter cette probabilité ?

(Indication : il existe une stratégie telle qu'ils aient une probabilité > 1-ln2 de s'en sortir. Ça parait vraiment surprenant mais c'est possible)

Le prisonnier spécialiste des probabilités a raison, de cette manière les chances de sorties sont infinitésimales.

Un étudiant d'ING1, très curieux, a décidé d'utiliser la POO afin de vérifier ces probabilités.

Il y a effectivement un moyen d'augmenter cette probabilité de sortie et nous allons le voir dans cet examen. Nous allons commencer dans la première partie par modéliser ce problème.

Partie 1

Nous allons dans cette première partie modéliser la classe *Coffre*. Il y aura 100 coffres numérotés de 0 à 99 (pour simplifier) et chaque coffre contiendra une valeur unique entre 0 et 99.

Comme cet étudiant est consciencieux, il a commencé à écrire le diagramme UML de cette classe.

Quelques informations sur ce diagramme de classe, le petit losange que l'on trouve dans les attributs *NB_PRISONNIERS, compteur, tableau, r* indiquent que ce sont des attributs de **classe** (mot-clef static) qui sont communs à toutes les instances de la classe, de même que la méthode *shuffle* qui sera une méthode de classe.

Le cadenas fermé indique que les attributs sont privés (mot-clef **private**), le cadenas ouvert indique que les attributs ou méthodes sont publics (mot-clef **public**).

L'attribut *compteur* sera initialisé à 0 (private static int *compteur* = 0) incrémenté de 1 pour chaque *coffre*. *NB_PRISONNIERS* aura la valeur 100. tableau est un attribut de classe de type tableau de 100 valeurs entières, le tableau sera utilisé pour contenir la valeur du coffre.

Par exemple, pour savoir ce que la valeur du coffre n°25, on ira voir ce que contient tableau[25]

Exercice :

Écrire la classe *Coffre* à partir du diagramme de classe et des indications suivante :

- lors de la création du premier coffre (c'est-à-dire lorsque l'attribut compteur vaut 0), dans le constructeur, vous initialiserez la variable *r* (type Random) et également le tableau de classe tableau de manière à ce que la ième cellule de i ait la valeur i *(tableau[i] = i)*

- le numéro de chaque *coffre* est passé dans le constructeur comme paramètre

- la méthode *getValeur()* vous retournera la valeur contenue dans le *coffre* (pour le moment le coffre i retournera la valeur i)

- la méthode de classe *shuffle()* permet de réaliser un mélange au sein des valeurs contenues dans notre coffre (attribut tableau). Vous pouvez pour mélanger ces valeurs en vous inspirant de la méthode du mélange vue dans le TP sur le blackjack.

- Faites une méthode *toString()* qui pour chaque coffre affiche son numéro et son contenu

Le programme suivant doit vous donner un certain nombre d'idées sur le fonctionnement. Vous **devez** reproduire ce fonctionnement avant de passer à la suite.

```
public class Main {

  public static void main(String [] args){


      // Premier test
      Coffre [ ] lesCoffres = new Coffre[100];

      for(int i = 0; i < Coffre.NB_PRISONNIERS; i++) {
          lesCoffres[i] = new Coffre(i);
          System.out.println(lesCoffres[i]);
      }

  }

}
```

Ce premier programme **doit** vous afficher :

Coffre{numéro=0, valeur=0}

Coffre{numéro=1, valeur=1}

Coffre{numéro=2, valeur=2}

Coffre{numéro=3, valeur=3}

Coffre{numéro=4, valeur=4}

Coffre{numéro=5, valeur=5}

Coffre{numéro=6, valeur=6}

Coffre{numéro=7, valeur=7}

Coffre{numéro=8, valeur=8}

Coffre{numéro=9, valeur=9}

Coffre{numéro=10, valeur=10}

Coffre{numéro=11, valeur=11}

….

Coffre{numéro=95, valeur=95}

Coffre{numéro=96, valeur=96}

Coffre{numéro=97, valeur=97}

Coffre{numéro=98, valeur=98}

Coffre{numéro=99, valeur=99}

Lorsqu'on introduit la méthode *shuffle()* qui mélange le tableau.

```
public class Main {

  public static void main(String [] args){

      // Premier test
      Coffre [ ] lesCoffres = new Coffre[100];

      for(int i = 0; i < Coffre.NB_PRISONNIERS; i++) {
          lesCoffres[i] = new Coffre(i);
      }

      // Deuxième test

      Coffre.shuffle();

      for(int i = 0; i < 100; i++) {
          System.out.println(lesCoffres[i]);
      }


  }

}
```

Vous devez obtenir une permutation dans les valeurs du tableau et chaque coffre contiendra une valeur différente :

Coffre{numéro=0, valeur=10}

Coffre{numéro=1, valeur=34}

Coffre{numéro=2, valeur=17}

Coffre{numéro=3, valeur=9}

Coffre{numéro=4, valeur=44}

Coffre{numéro=5, valeur=11}

Coffre{numéro=6, valeur=53}

Coffre{numéro=7, valeur=56}

Coffre{numéro=8, valeur=13}

...

...

Coffre{numéro=96, valeur=1}

Coffre{numéro=97, valeur=46}

Coffre{numéro=98, valeur=21}

Coffre{numéro=99, valeur=84}

Partie 2

Nous allons maintenant continuer en introduisant les prisonniers et chaque prisonnier essayera à tour de rôle d'ouvrir 50 coffres au hasard. S'il trouve, dans un des 50 coffres qu'il ouvre son numéro, on pourra considérer qu'il peut arrêter. Mais pour que tous les prisonniers sortent, il faut que les 100 prisonniers trouvent leur numéro, ce qui est pratiquement impossible. Nos verrons une stratégie en partie 3 qui leur permet de sortir avec une probabilité de 30 %.

Notre étudiant d'ING1 toujours consciencieux a commencé à écrire le diagramme de classe de la classe Prisonnier.

Écrire la classe *Prisonnier* à partir du diagramme de classe et des indications suivante :

- le constructeur de la classe *Prisonnier* prend en paramètre comme dans le cas de la classe *Coffre* le *numéro* de prisonnier.
- l'attribut *sortie* de type booléen indique si ce prisonnier a réussi à sortir (true) et est positionné à faux (false)
- la méthode *getNuméro()* retourne bien entendu le numéro du prisonnier
- la méthode *ouvreCoffres* prend en paramètre un tableau de 100 coffres (*cf* plus loin) et essaye d'en ouvrir 50 au hasard. Si en ouvrant 50 coffres, le prisonnier arrive à trouver son numéro dans un des coffres, l'attribut sortie passera à vrai (true)
- la méthode *setSortie* ne fait que positionner à vrai ou faux l'attribut *sortie*. L'utilité de cette méthode apparaîtra quand on utilisera l'héritage
- la méthode *toString* affiche simplement le numéro du prisonnier et s'il a réussi à sortir.

Le programme main suivant vous permettra de tester votre classe *Prisonnier* et l'utilisation de la méthode ouvre coffre *ouvreCoffres* . Il doit être lancé bien évidemment après l'initialisation des coffres !

```
// troisième test

    Prisonnier [] lesPrisonniers = new Prisonnier[100];

    for(int i = 0; i < 100; i++){
        lesPrisonniers[i] = new Prisonnier(i);
    }


    for(int i = 0; i < 100; i++){
        lesPrisonniers[i].ouvreCoffres(lesCoffres);
        System.out.println(lesPrisonniers[i]);
```

```
    }
```

Prisonnier{numéro=0, sortie=false}

Prisonnier{numéro=1, sortie=false}

Prisonnier{numéro=2, sortie=true}

Prisonnier{numéro=3, sortie=false}

...

Prisonnier{numéro=96, sortie=true}

Prisonnier{numéro=97, sortie=false}

Prisonnier{numéro=98, sortie=true}

Prisonnier{numéro=99, sortie=true}

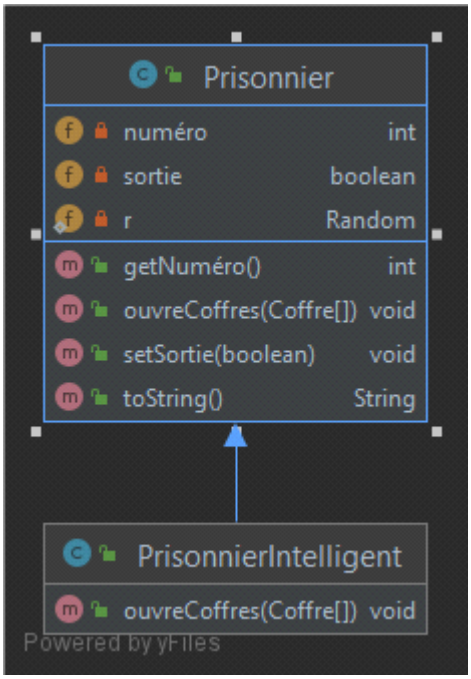
Normalement, vous devez constater que grosso modo, 50 % des prisonniers arrivent à trouver leur numéro mais pour sortir, il faudrait que tous les prisonniers trouvent leur numéro, ce qui est impossible avec cette méthode. C'est ce que nous allons voir dans cette troisième partie

Partie 3

Dans cette dernière partie, nous allons étudier une nouvelle stratégie permettant de faire sortir les prisonniers. La stratégie qui permet de sortir avec une probabilité de 30 % est très simple chaque prisonnier démarre par le coffre portant son numéro. Soit ce coffre contient le numéro du prisonnier et dans cas, le prisonnier arrête soit il ne contient pas son numéro et le prisonnier va ouvrir le coffre dont le numéro est celui qu'il vient de trouver dans le coffre.

Par exemple, le prisonnier 68 va ouvrir le coffre n°68, il trouve le numéro 29, il va ouvrir le coffre n°29 et ainsi de suite jusqu'au maximum de 50 coffres. Si dans un des coffres, il trouve son numéro, il s'arrête et positionne l'attribut *sortie* à vrai (true)

Notre étudiant d'ING1 qui a bien suivi les cours décide d'utiliser l'héritage et de créer une classe *PrisonnierIntelligent* qui héritent de la classe *Prisonnier* et qui va redéfinir la méthode *ouvreCoffres()* en utilisant la stratégie décrite ci-dessus.



A vous d'écrire cette classe. Vous pouvez vérifier le fonctionnement de votre nouvelle classe avec le programme suivant, vous devriez sur plusieurs exécutions constater des sorties complètes de tous les prisonniers

// troisième test pour la classe *PrisonnierIntelligent*

```
Prisonnier [] lesPrisonniersIntelligents = new PrisonnierIntelligent[100];

        for(int i = 0; i < 100; i++)
            lesPrisonniersIntelligents[i] = new PrisonnierIntelligent(i);



        for(int i = 0; i < 100; i++){
            lesPrisonniersIntelligents[i].ouvreCoffres(lesCoffres);
            System.out.println(lesPrisonniersIntelligents[i]);
        }
```

Prisonnier{numéro=0, sortie=true}

Prisonnier{numéro=1, sortie=true}

Prisonnier{numéro=2, sortie=true}

Prisonnier{numéro=3, sortie=true}

Prisonnier{numéro=4, sortie=true}

Prisonnier{numéro=5, sortie=true}

...

Prisonnier{numéro=97, sortie=true}

Prisonnier{numéro=98, sortie=true}

Prisonnier{numéro=99, sortie=true}